React

Shan-Hung Wu & DataLab
CS, NTHU

Separation of Concerns

- HTML: noun
- CSS: adjective
- JS: verb
- Applies well to ***static*** pages
- But for ***dynamic*** content, JS usually “crosses the line”

```
el.textContent = '...';
```

Modern Frontend Framework

- Write JS ***in HTML***

- Write HTML ***in JS***

React

- Component-based
- Debug-friendly
  - Syntax errors fails at compile time
  - ***Declarative*** properties make runtime debugging easy
- Extends to mobile landscape (React Native)

Hello React

```
$ npm install --save react react-dom
$ npm install --save-dev babel-preset-react

// in webpack.config.js
entry: {
  index: './index.jsx',
  vendor: ['react', 'react-dom']
}
module: {
  rules: [{
    test: /\.(js|jsx)$/,
    exclude: [/node_modules/],
    use: [{
      loader: 'babel-loader',
      options: {
        presets: [
          ['es2015', {'modules': false}],
          'react'
        ]
      }
    }]
  }], ...
}
```

- `*.jsx` are JS files with HTML embedded

Webpack Aliasing

- Components are move to `src/components`
  - `Component.jsx` by convention
- Use aliasing to resolve path dependency:

```
// under "module.exports" in webpack.config.js
resolve: {
  alias: {
    components: path.resolve(srcPath, 'components')
  }
}, ...

// in index.js
import ... from 'components/Component.jsx';
```

Development Server

```
$ npm install --save-dev \
  webpack-dev-server

// in webpack.config.js
module.exports = {
  ...,
  devServer: {
    contentBase: distPath,
    compress: true,
    port: 8080
  }
}

// under "scripts" in package.json:
"start": "webpack-dev-server",
```

- React requires a web server to run

Outline

- React components and JSX
  - States and data flows
  - Forms
  - More JSX
- Thinking in React: WeatherMood
  - Routing
  - Promises and AJAX requests
  - UI
- Debugging

Outline

- React components and JSX
  - States and data flows
  - Forms
  - More JSX
- Thinking in React: WeatherMood
  - Routing
  - Promises and AJAX requests
  - UI
- Debugging

Hello React

```
// in index.js
import React from 'react';
import ReactDOM from 'react-dom';

window.onload = function() {
  let name = 'Bob';
  ReactDOM.render(
    <h1>Hello {name}</h1>, // JSX, no quotes
    document.getElementById('root')
  );
};
```

- ***JSX*** allows writing HTML in JS
- Compiled to normal ***objects*** by Babel

```
const el = <h1>Hello {name}</h1>;
// compiled to
const el = React.createElement('h1', ...);
```

JSX Basics

```
// embedded expressions
const el = <div>Number {1+1}</div>;

// multi-line with nested elements
const el = (
  <div>
    <h1>Title</h1>
    <p>Paragraph.</p>
  </div>
); // use (...) to prevent auto ';' insertion

// attributes (in lower-camel-case)
const url = '...';
const el = <img className='fliud' src={url}>;
```

- Everything is converted to a string before rendered
- Prevents injection attacks, e.g., cross-site-scripting (XSS)

Fast Re-rendering

- React keeps a virtual DOM in memory
  - Updates only ***changed elements*** in real ROM

```
function tick() {
  const date = new Date().toLocaleTimeString();
  const el = (
    <div>
      <h1>Hello</h1>
      <h2>It's {date}.</h2>
    </div>
  );
  ReactDOM.render(el, document.getElementById('root');
}
setInterval(tick, 1000);
```

Components & Props

```
// functional component
function Component(props) {
  return <h2>This is {props.name}<h2>;
}
// class component
class Component extends React.Component {
  constructor(props) {
    super(props); ...
  }
  render() {
    return <h2>This is {this.props.name}<h2>;
  }
}
// in index.js
ReactDOM.render(
  <div>
    <Component name='Alice'>
    <Component name='Bob'>
  </div>,
  document.getElementById('root');
);
```

- `render()` accepts only a single root element
- Multiple instances created

Never change props in a component!

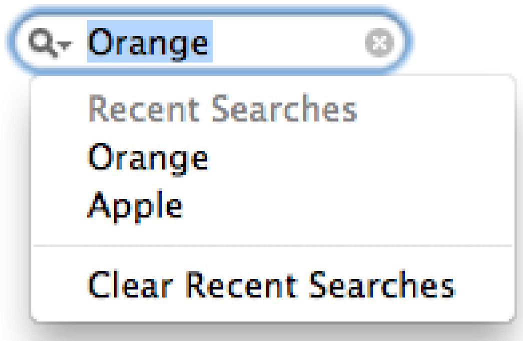
- So, React can efficiently detect whether a component should be re-rendered

Outline

- React components and JSX
  - States and data flows
  - Forms
  - More JSX
- Thinking in React: WeatherMood
  - Routing
  - Promises and AJAX requests
  - UI
- Debugging

States

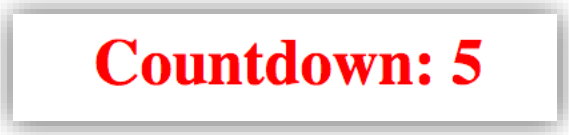
- A component may have its own ***states***
- Keep track of e.g., user input or program status



Example: Counter

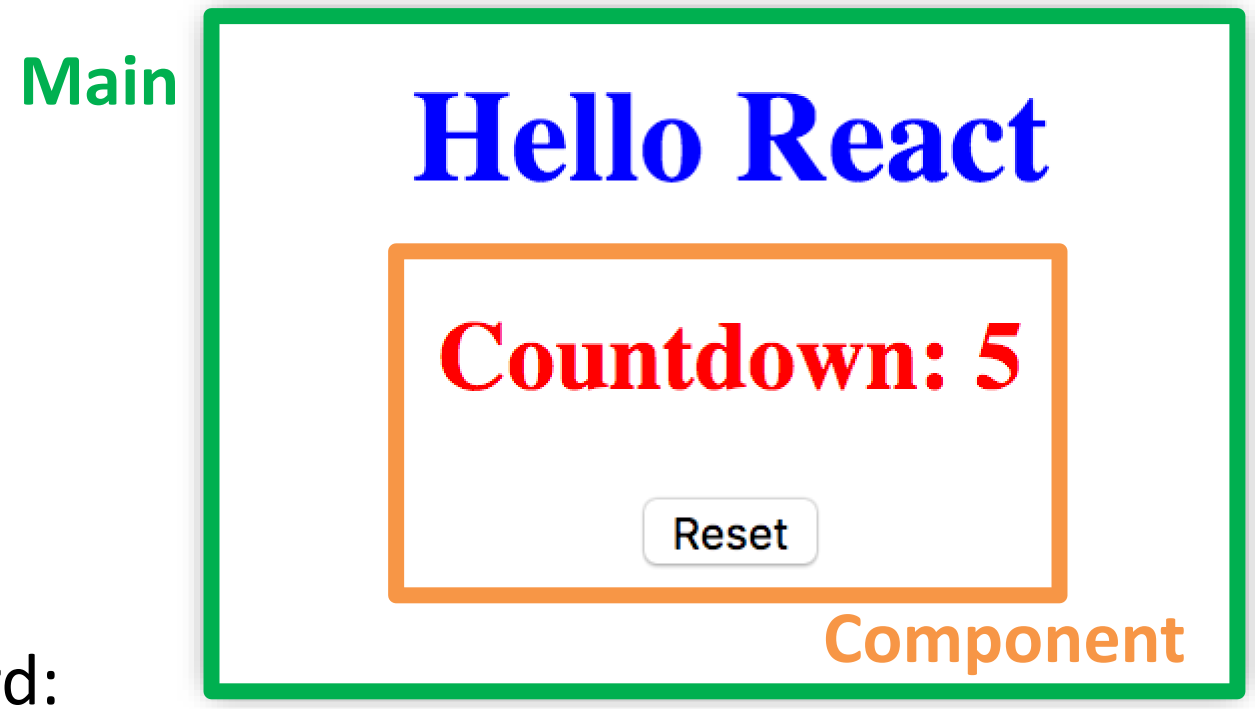
```
class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 5
    };
  }
  render() {
    return <h2>Countdown: {this.state.count}</h2>;
  }

  // lifecycle methods
  componentDidMount() {
    this.countdownId = setInterval(() => {
      this.setState({ // triggers re-rendering
        count: this.state.count - 1
      }); // obj merged to this.state
    }, 1000);
  }
  componentWillUnmount() {
    clearInterval(this.countdownId);
  }
}
```


Countdown: 5

- More lifecycle methods

Data Flows

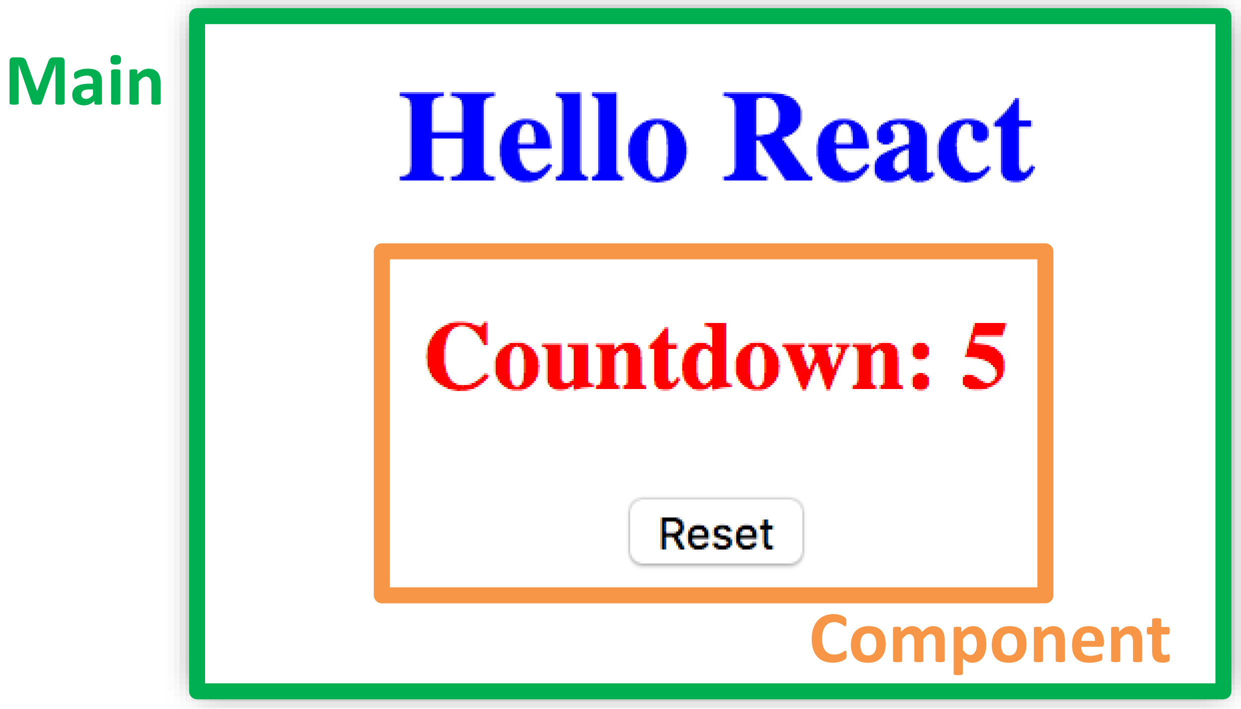


- Downward:

```
// in Main.render()
<Component count={this.state.count}>

// in Component.render()
<h2>Countdown: {this.props.count}</h2>
```

Data Flows



- Upward:

```
// in Main.render()
<Component onReset={this.handleReset}>

// in Component.render()
<button onClick={this.props.onReset}>Reset</button>
```

Advantages

- Data flows are ***declarative***
  - JSX declares both downward and upward flows
  - Simplifies debugging (by investigating `props`)
- States are local to component ***objects***
  - Simplifies state management in complex UIs

```
// in index.js
ReactDOM.render(
  <div>
    <Main />
    <Main />
  </div>,
  document.getElementById('root');
);
```

Bug 1: `this` in `handleReset()`

```
// in Main.render()
<Component onReset={this.handleReset}>

// in Component.render()
<button onClick={this.props.onReset}>Reset</button>
```

- `this` does not bind to `Main` when called
- Fix:

```
// in Main.constructor()
this.handleReset = this.handleReset.bind(this);

// or use ES7 property initializer
class Main extends React.Component {
  handleReset = () => {
    ...
  };
}
```

Bug 2: `setState()` is Asynchronous

- React batches multiple `setState` calls for better rendering performance

```
setState({
  count: this.state.count - 1
});
... // uses new count
```

- If new state depends on previous one (or props):

```
setState((prevState, props) => ({
  count: prevState.count - 1
}));
```

- Actions assuming new state is set:

```
setState({
  count: prevState.count - 1
}, () => {
  ... // uses new count
});
```

Outline

- React components and JSX
  - States and data flows
  - Forms
  - More JSX
- Thinking in React: WeatherMood
  - Routing
  - Promises and AJAX requests
  - UI
- Debugging

Form Elements

- Form elements manages their own state in normal JS
- ***Uncontrolled*** form elements:

```
// JSX in render()
<form>
  <input type='text' ref={el => {this.inputEl = el}} />
</form onSubmit={this.handleSubmit}>

// method
handleSubmit(e) {
  const v = this.inputEl.value;
  this.inputEl.blur();
  ...
}
```

Controlled Form Elements

- States managed explicitly (recommended)
  - More declarative in JSX

```
// JSX in render()
<form>
  <input type='text' value={this.state.inputValue}
      onChange={this.handleInputChange} />
</form onSubmit={this.handleSubmit}>

// methods
handleInputChnage(e) {
  this.setState({inputValue: e.target.value});
}
handleSubmit(e) {
  const v = this.state.inputValue;
  ...
}
```

Outline

- React components and JSX
  - States and data flows
  - Forms
  - More JSX
- Thinking in React: WeatherMood
  - Routing
  - Promises and AJAX requests
  - UI
- Debugging

Conditions

- Only expressions in `{}`: no `if`, `for`, etc.
- `{}` must be evaluated to DOM element(s)

```
// if
<h1>Hello</h1> {
  unreadMsgs.length > 0 && // short circuit
      <h2>You have {unreadMsgs.length} messages.</h2>
}
// if else
<h1>Hello</h1> {
  unreadMsgs.length == 0 ? <h2>No message.</h2> :
      <h2>You have {unreadMsgs.length} messages.</h2>
}
```

Loops

```
// for
<h1>Unread messages:</h1>
<ul> {
  unreadMsgs.map((m => <li key={m.id}>{m.title}</li>))
}
</ul>
```

- Arrays/lists of elements can be rendered directly
- Key per item is necessary for fast re-rendering
  - Only needs to be unique among siblings
  - Added to the composing component (that calls `map`), rather than individual element components

Attributes

- Default to true

```
<Message isRead />  // same as
<Message isRead={true} />
```

- Spread

```
const props = {name: 'Bob', age: 18}
<User {...props} />
```

Composition

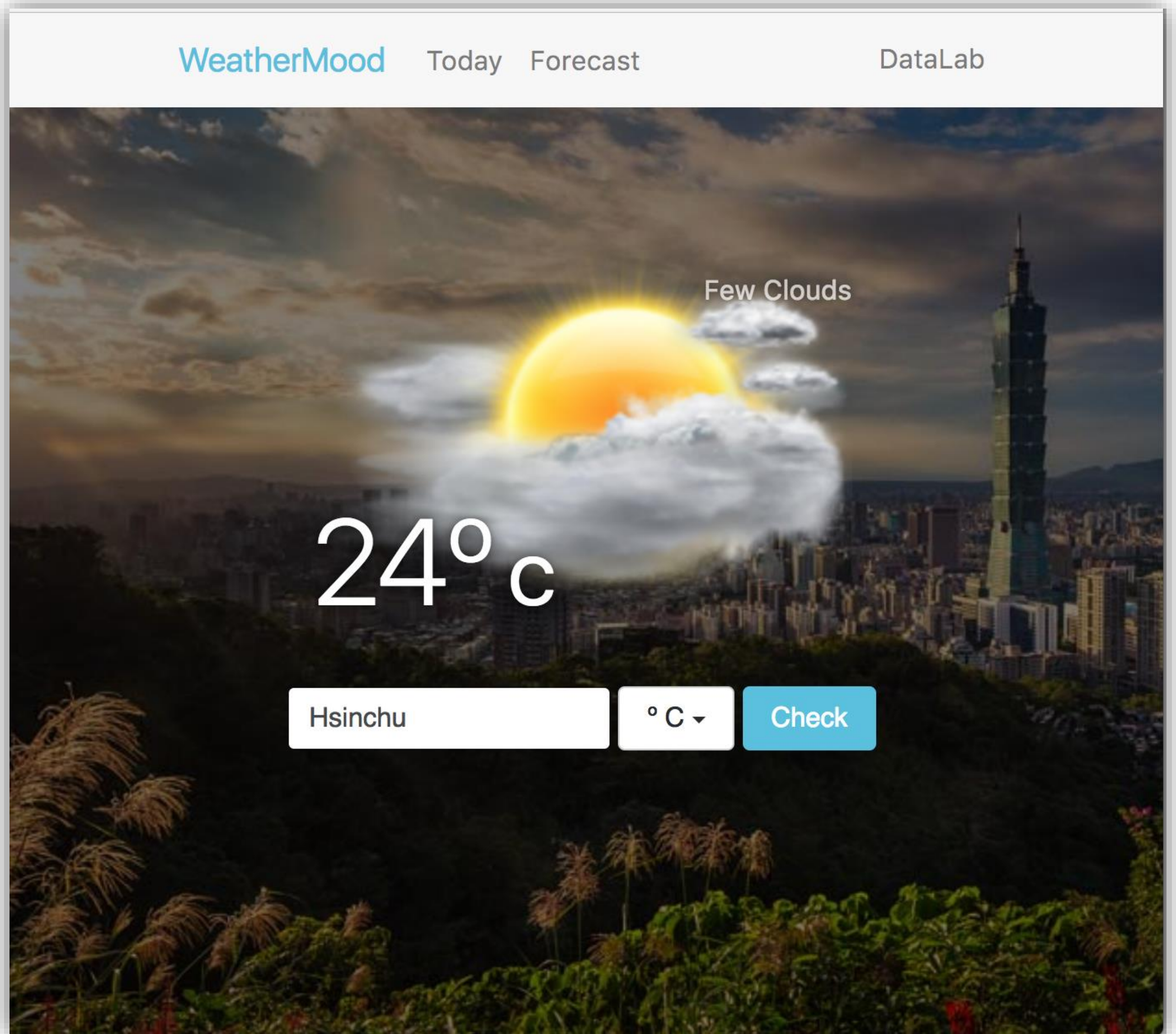
```
function GeneralComponent {
  return (
    <div>
      <h1>Welcome</h1>
      {props.children} // provided by React
    </div>
  );
}
function SpecializedComponent {
  return (
    <GeneralComponent>
      <p>Content</p>
      <p>Footer</p>
    </GeneralComponent>
  );
}
```

- Facebook favors composition over inheritance

Outline

- React components and JSX
  - States and data flows
  - Forms
  - More JSX
- Thinking in React: WeatherMood
  - Routing
  - Promises and AJAX requests
  - UI
- Debugging

Clone `lab-react-weathermood`



Setup

```
$ npm install --save react-router axios
$ npm install --save reactstrap \
  react-addons-transition-group \
  react-addons-css-transition-group
```

- React Router
  - Loads different components based on different URL paths
  - Declarative (in JSX)
- Axios
  - Makes AJAX (in-page HTTP) requests
  - ES6 Promise-based API
- Reactstrap
  - Bootstrap with JS replaced by React
  - Uses React animation add-ons

ES7 Spread in Object Literals

```
$ npm install --save-dev \
  babel-plugin-transform-object-rest-spread

// in webpack.config.js
loader: 'babel-loader',
options: {
  presets: [...],
  plugin: [..., 'transform-object-rest-spread']
}

// merge objects
const obj1 = {
  p1: 1,
  p2: 2
};
const obj2 = {
  ...obj1,
  p3: 3
}; // obj2 now has p1, p2, and p3
```

Turning off URL Translation in CSS

- By default, Babel CSS-loader translates `url(…)`'s into module imports
- Problematic when setting, e.g., background images

```
background-image: url('...');
```

- To turn this off:

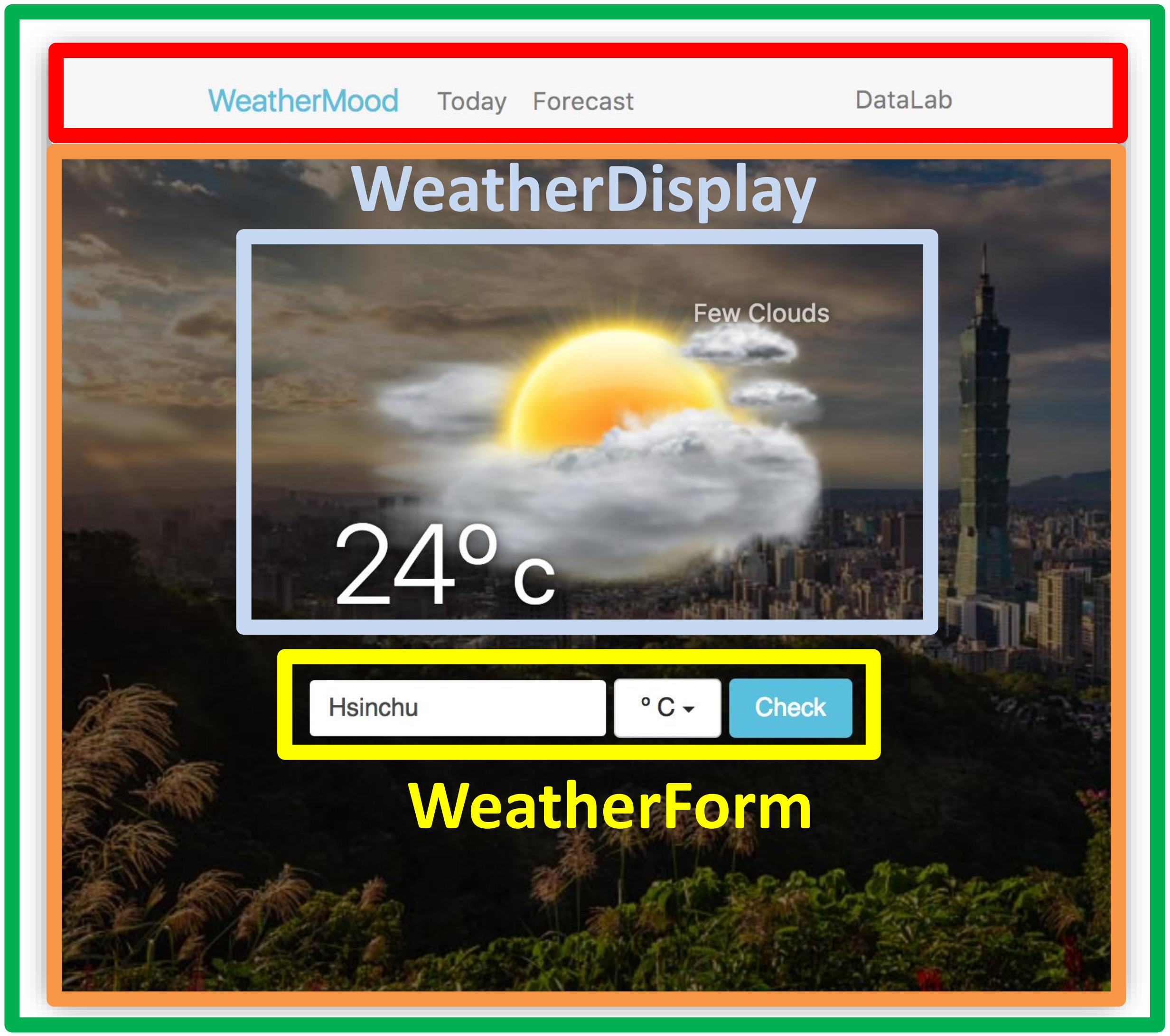
```
// in rules of webpack.config.js
test: /\.css$/,
use: ['style-loader', {
  loader: 'css-loader',
  options : {
    url: false
  }
}]
```

Open
Weather
Map

- Sign up to get an App ID

Thinking in React

Step 1: Component Hierarchy

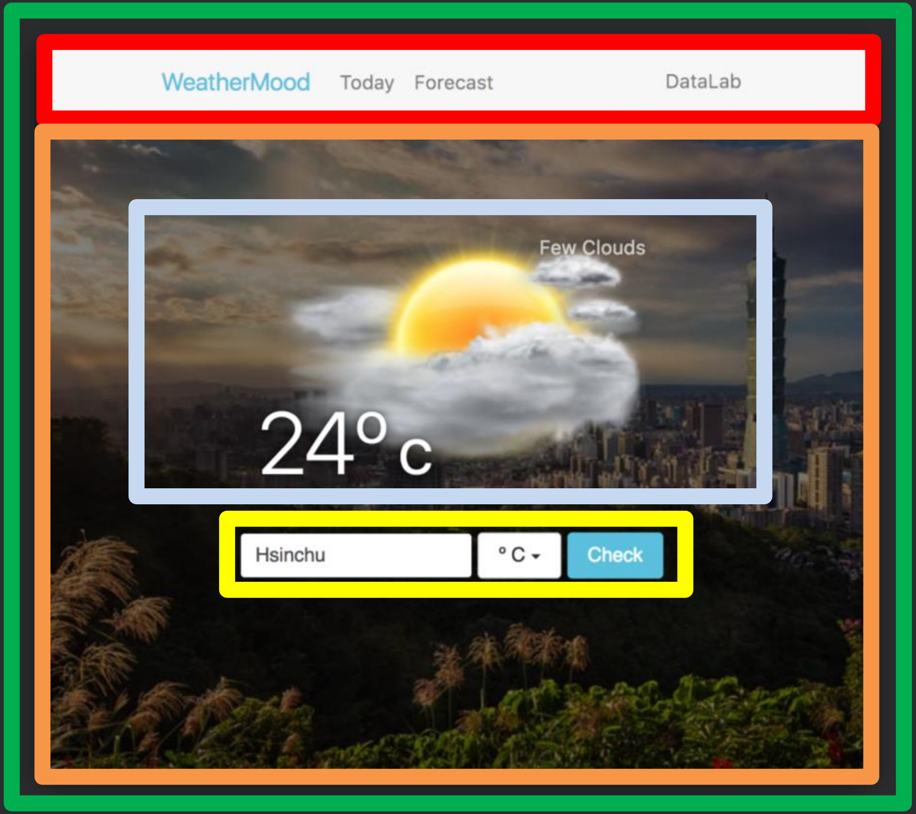


Navbar

Today / Forecast

Main

Step 2: Static Version (Downward Data Flow)



Main

Navbar { link1, link2 }

WeatherDisplay {
  temp, unit
  weather, desc
}

WeatherForm {
  city, unit
}

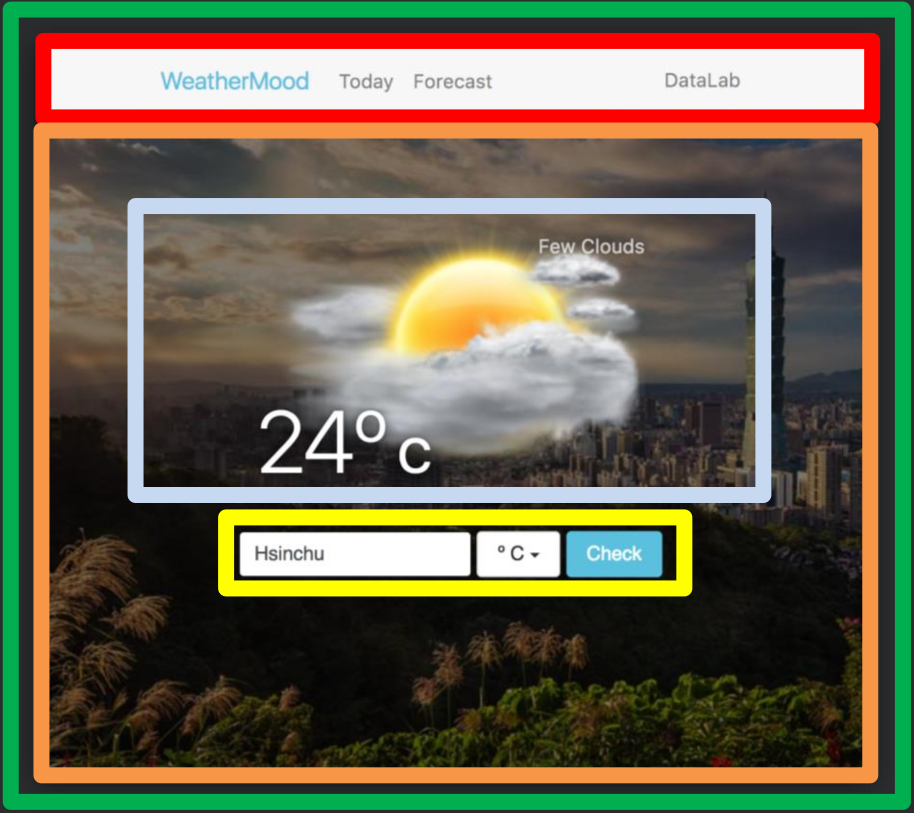
Today { weather }
Forecast { weather }

Step3: Identifying States

- The “changing parts” of a component
  - By user or program logic

- Passed in from a parent via props?
  - Not a state
- Remain unchanged over time?
  - Not a state
- Can be computed based other states/props?
  - Not a state

Step 3: Identifying States



Main

Navbar { ~~link1~~, ~~link2~~ }

WeatherDisplay {
  temp, unit
  ~~weather~~, desc
}

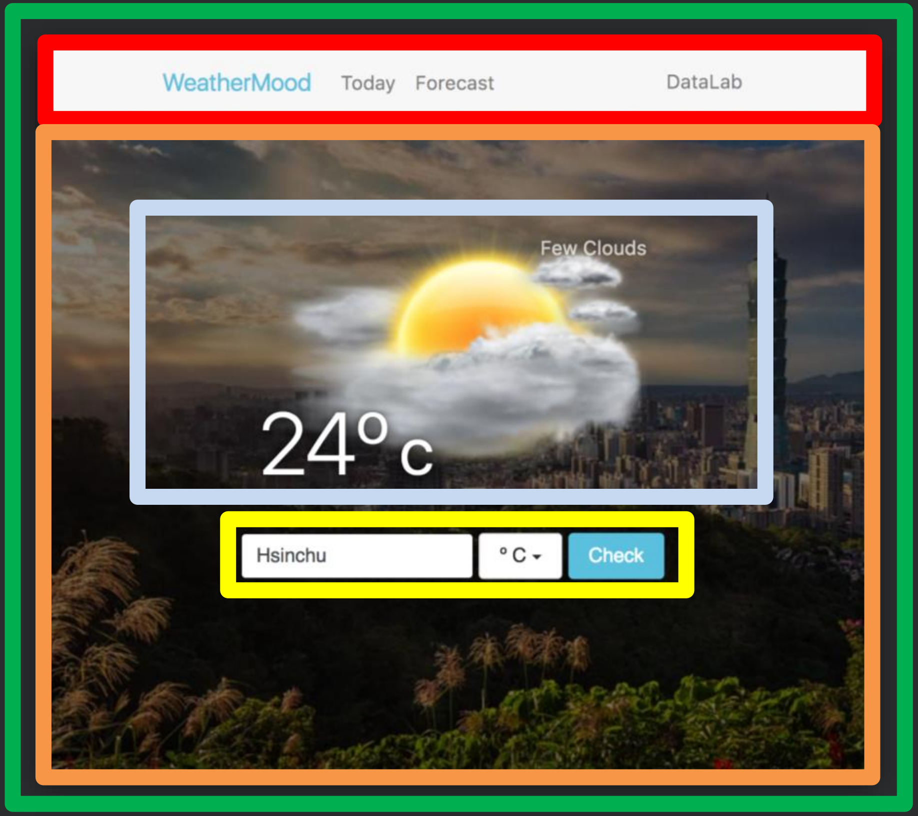
WeatherForm {
  city, unit
}

Today { weather }
Forecast { weather }

Step 4: Lifting States Up

- What if several components need to reflect the same/corresponding states?
- Lift the states up to the ***closest common ancestor***

Step 4: Lifting States Up



Main {
unit
}

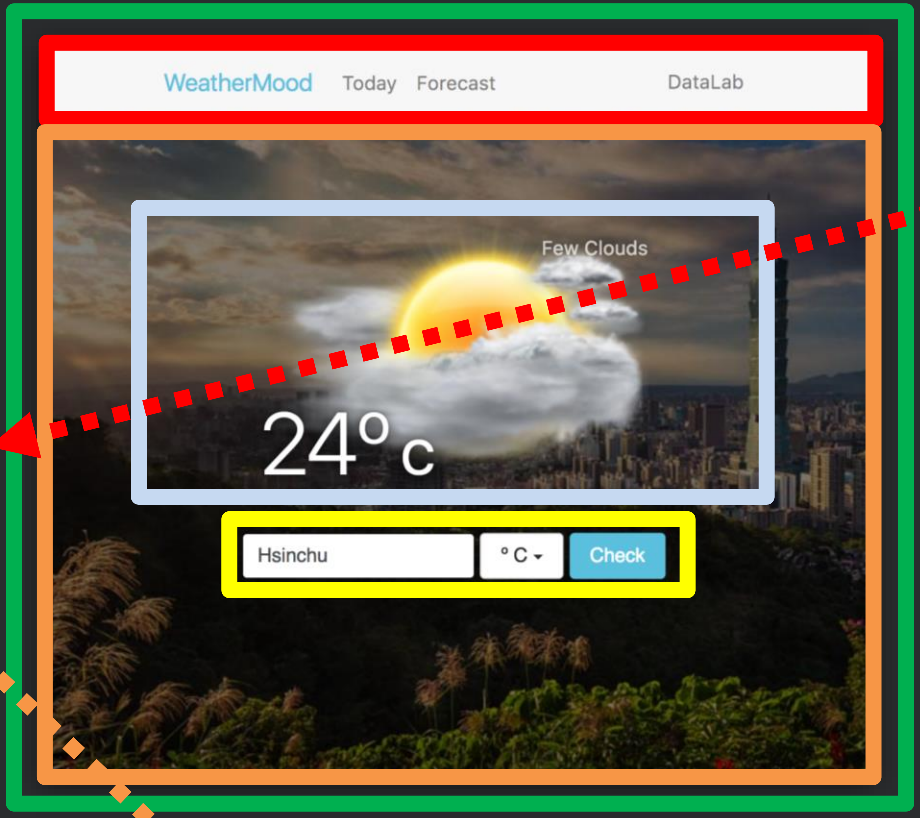
Navbar { ~~link1~~, ~~link2~~ }

WeatherDisplay {
~~temp~~, ~~unit~~
~~weather~~, ~~desc~~
}

WeatherForm {
~~city~~, ~~unit~~
}

Today { weather, temp, desc, city }
Forecast { weather, temp, desc, city }

Step 5: Upward Data Flow



Navbar { ~~link1~~, ~~link2~~ }

WeatherDisplay {
~~temp~~, ~~unit~~
~~weather~~, ~~desc~~
}

WeatherForm {
~~city~~, ~~unit~~
}

Main {
unit
}

Today { weather, temp, desc, city }
Forecast { weather, temp, desc, city }

Outline

- React components and JSX
  - States and data flows
  - Forms
  - More JSX
- Thinking in React: WeatherMood
  - Routing
  - Promises and AJAX requests
  - UI
- Debugging

Client-Side Routing

- A single-page app may want to load different components based on different URL paths
  - At client side to minimize latency
- Implementation: a `Link` component that
  - Fires an event when clicked
  - Tells which path to go
- Container component can then mount/unmount relevant components

React Router

- Declarative client-side routing:

```
// in Main.jsx
render() {
  return (
    <Router>
      <Link to='/'>Today</Link>
      <Link to='/forecast'>Forecast</Link>
      ...
      <Route exact path='/'
          render={() => <Today />}></Route>
      <Route path='/forecast'
          render={() => <Forecast />}></Route>
    </Router>
  );
}
```

- More matching rules

Outline

- React components and JSX
  - States and data flows
  - Forms
  - More JSX
- Thinking in React: WeatherMood
  - Routing
  - Promises and AJAX requests
  - UI
- Debugging

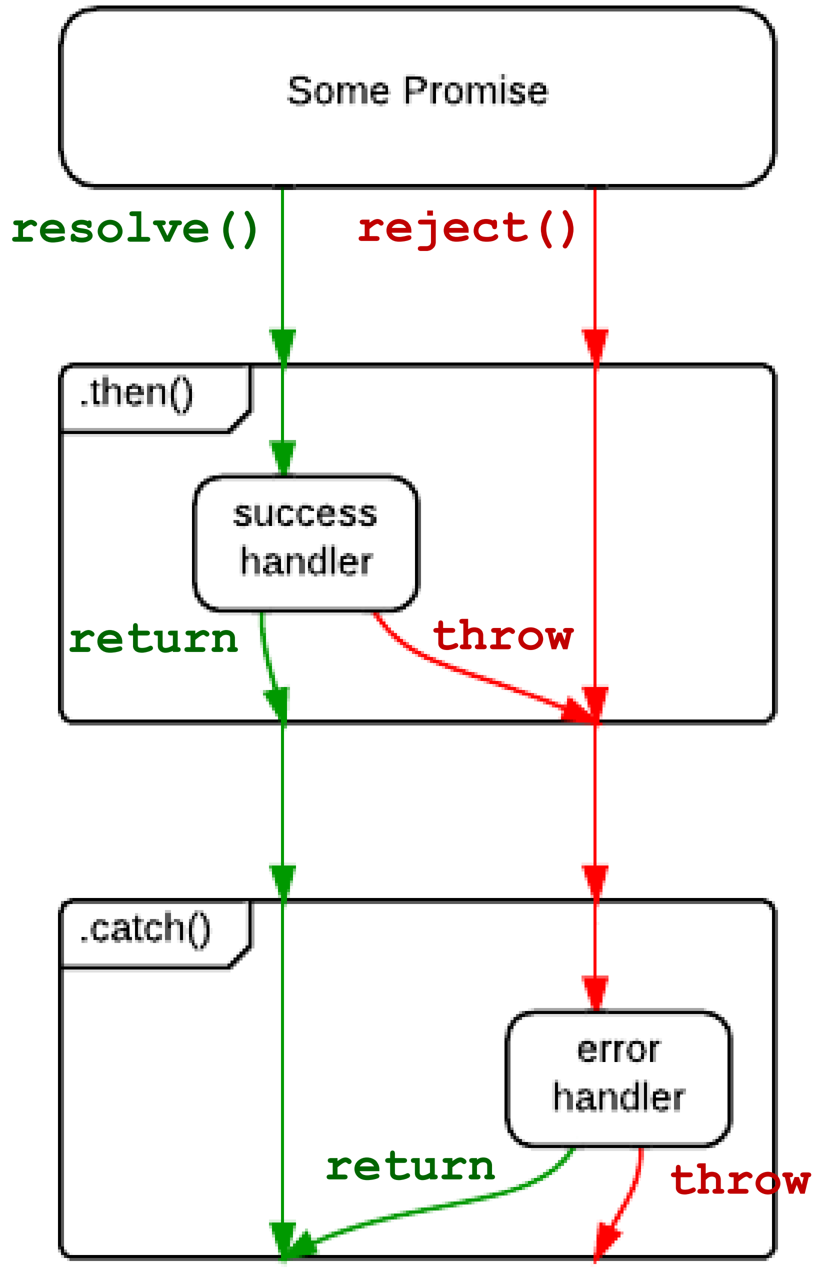
ES6 Promise

```
// in method1()
const p = new Promise((resolve, reject) => {
  ... // do asynchronous job here
  if (success) resolve(data);
  else reject(err);
});
return p;

// in method2(p)
const p2 = p.then(data => {
   ... // process data
  return data2
}); // always returns a new Promise
return p2;

// in method3(p2)
p2.then(data2 => {
  ... process data2
}).catch(err => {
  ... // handle err
}); // always returns a new Promise
```

- A value available ***in the future***
- Separation of concerns
  - Handlers can be written in different places
- Use arrow func for `this`



Execution Flow

- Chain `then` and/or `catch` as long as you like
- Reject mode:
  - `throw new Error()`
- Resolve mode:
  - `return`

Axios and AJAX Requests

```
// GET request
axios.get('...url...').then(res => {
  res.status  // HTTP response code (e.g., 200, 401)
  res.data    // object parsed from HTTP response body
  res.headers // HTTP presonse headers
}).catch(err => {
  console.log(err);
});

// POST request
axios.post('...url...', {
  ... // request body
}).then(...).catch(...);
```

- Requests can be canceled

Outline

- React components and JSX
  - States and data flows
  - Forms
  - More JSX
- Thinking in React: WeatherMood
  - Routing
  - Promises and AJAX requests
  - UI
- Debugging

Reactstrap

- Bootstrap with JS replaced by React
  - Integrates to virtual DOM for better performance

```
import {Button} from 'reactstrap';

// JSX
<Button color='primary'
    onClick={this.handleClick}>primary</Button>

handleClick(e) {
  ...
}
```

- More [components]

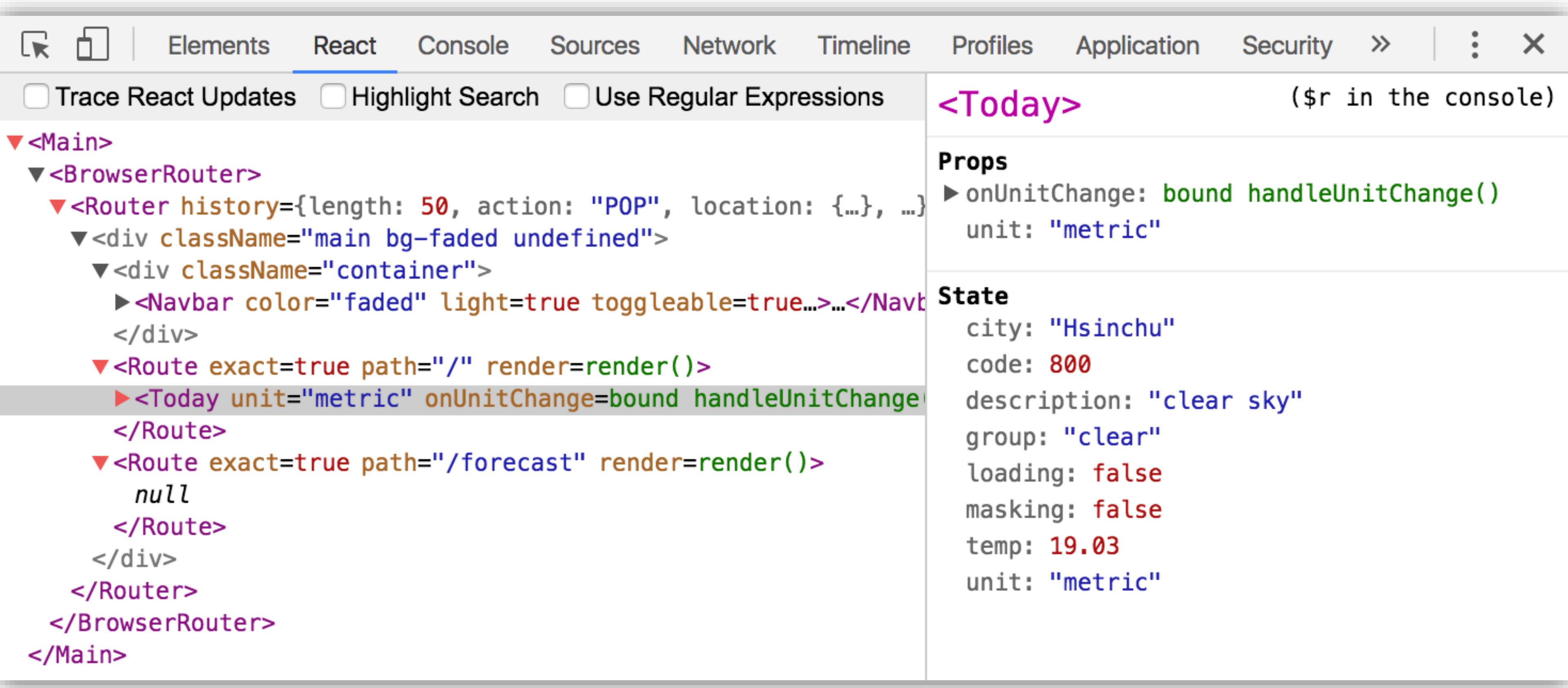
More React UI Libraries

- Material-UI
- React-Bootstrap
- and more

Outline

- React components and JSX
  - States and data flows
  - Forms
  - More JSX
- Thinking in React: WeatherMood
  - Routing
  - Promises and AJAX requests
  - UI
- Debugging

React DevTools



```
▼<Main>
  ▼<BrowserRouter>
    ▼<Router history={length: 50, action: "POP", location: {…}, …}
      ▼<div className="main bg-faded undefined">
        ▼<div className="container">
          ▶<Navbar color="faded" light=true toggleable=true…>…</Navb
          </div>
        ▼<Route exact=true path="/" render=render()>
          ▶<Today unit="metric" onUnitChange=bound handleUnitChange(
          </Route>
        ▼<Route exact=true path="/forecast" render=render()>
            null
          </Route>
        </div>
      </Router>
    </BrowserRouter>
</Main>
```

```
<Today>                    ($r in the console)

Props
▶ onUnitChange: bound handleUnitChange()
  unit: "metric"

State
  city: "Hsinchu"
  code: 800
  description: "clear sky"
  group: "clear"
  loading: false
  masking: false
  temp: 19.03
  unit: "metric"
```

- Breakpoints: `debugger;`

Type Checking

```
class WeatherForm extends Ract.Component {
 // ES7 property initializer
  static propTypes = {
    city: React.PropTypes.string,
    unit: React.PropTypes.string

    // bool, number, array, object, func, ...

  };

  constructor(props) {...}
  render() {...}
}
```

- More type checking

Source Map

```
// in webpack.config.js
module.exports = {
  ...
  devtool: 'source-map'
};
```

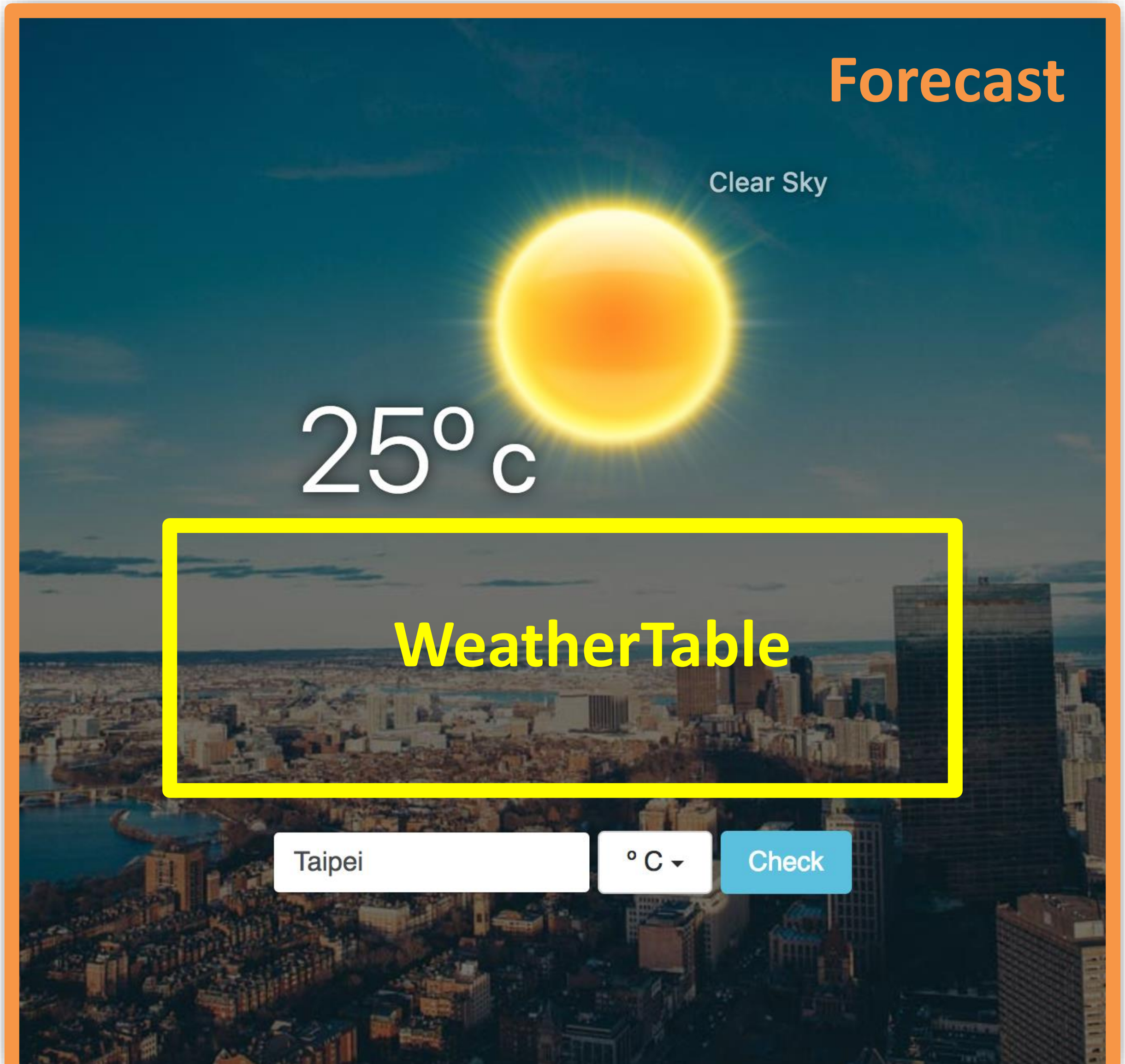
- Options & speed
- Usually, `'cheap-source-map'` is a good compromise

Assigned Readings

- Tic Tac Toe in React

- JSX in depth (optional)
- Controlled form elements (optional)
- Refs and uncontrolled components (optional)

Assignment: 5-Days Forecast



Requirements

- Use the 5-days forecast API
- Show “tomorrow” in `WeatherDisplay`
- `WeatherTable` for the rest days
- Each day in `WeatherTable`:
  - Day of week: “Sat,” “Sun ,” “Mon,” and so on
  - Temperature in correct unit
  - Weather condition icon (based on code)
- Responsive: show only 2 days in `WeatherTable` on portrait mobile devices

Bonus

- Query OWM using users' current geo-coordinates (latitudes and longitudes):

```
navigator.geolocation
    .getCurrentPosition(function(position) {
  // called back asynchronously
  const lat = position.coords.latitude;
  const lng = position.coords.longitude; ...
})
```

  - Read this tutorial first
  - Watch out the privacy settings
- Change background to Google Map, and allow “pin to query”
  - You can use react-google-maps or similar